Distribuované databázy (pokračovanie)

- Výber koordinátora
- Replikácia dát
  - Distribuovaný locking
  - Distribuovaná správa deadlockov
- Distribuované časové pečiatky (time-stamps)
- Synchronizácia času

Literatúra:

P.A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems, http://research.microsoft.com/pubs/ccontrol/

H. Garcia-Molina, J.D. Ullman, J. Widom: Database System Implementation, Prentice Hall, 2000

A.S. Tanenbaum: Distributed Operating Systems, Prentice Hall, 1995

Výber koordinátora

Pravidlá hry:

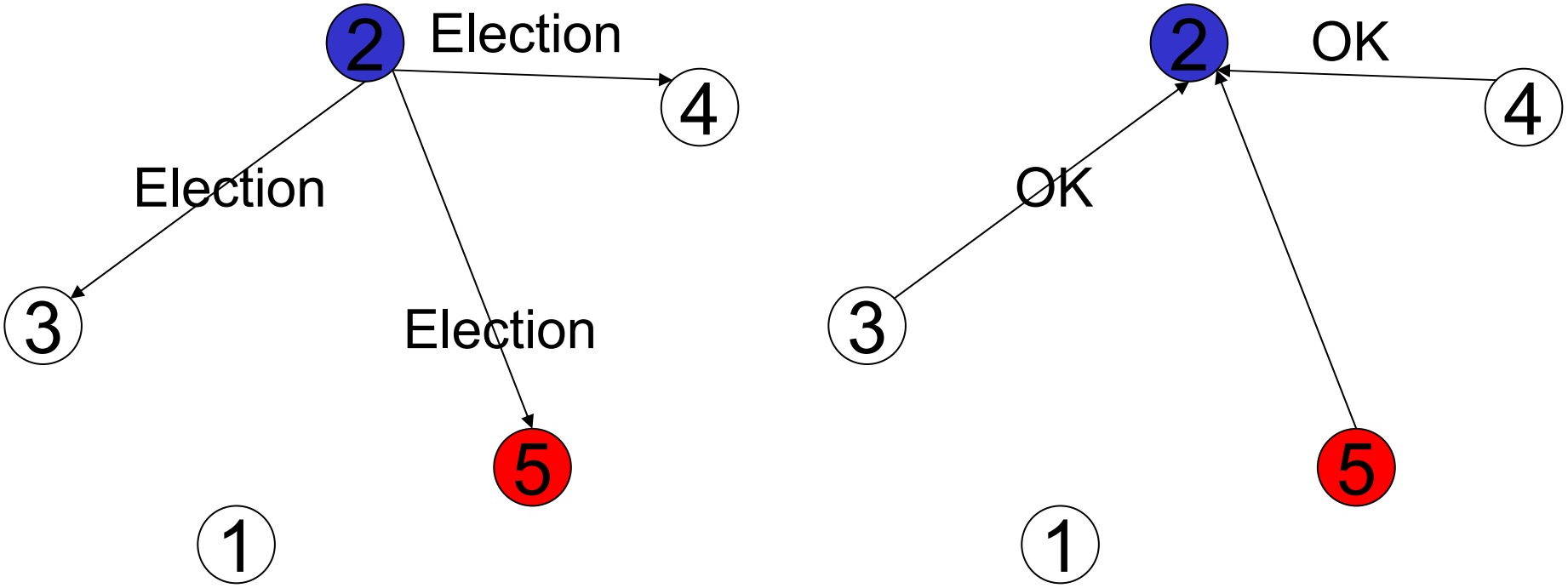
- Každý uzol má jednoznačný identifikátor
- Každý uzol pozná všetky identifikátory
- Komunikačný graf je v prípade výpadkov uzlov či liniek súvislý

Cieľ: Koordinátorom bude **nespadnutý** uzol s najväčším identifikátorom

Výber koordinátora

Bully algoritmus

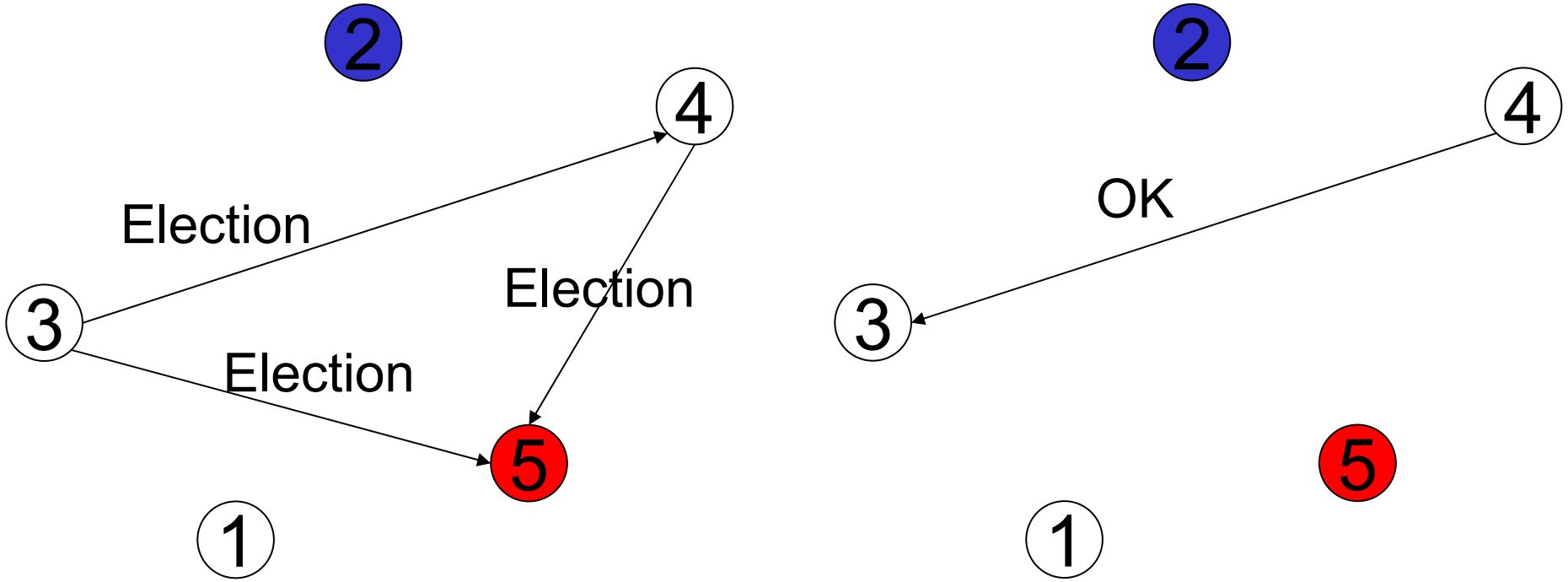
- Príklad: uzol 2 je iniciátor, uzol 5 je spadnutý



Výber koordinátora

Bully algoritmus

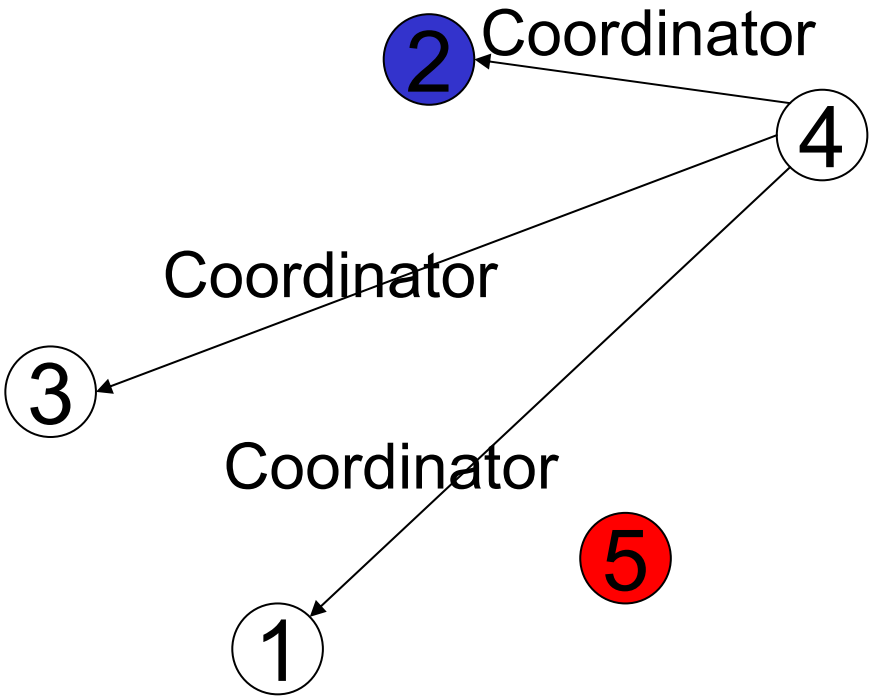
- Príklad: uzol 2 je iniciátor, uzol 5 je spadnutý



Výber koordinátora

Bully algoritmus

- Príklad: uzol 2 je iniciátor, uzol 5 je spadnutý



Replikácia dát

Pravidlá hry:

- Rovnaké dáta sú replikované vo viacerých uzloch
- Čítanie dát (read) je možné z ľubovoľného z týchto uzlov, ale prepisovanie dát (write) treba urobiť vo všetkých kópiách

Cieľ: Zabezpečiť sériovateľnosť (izoláciu) transakcií
→ distribuované zamykanie, distribuované časové pečiatky

Distribuované zamykanie (locking)

Centralizovaná schéma

- Jeden uzol je lock manager pre všetky transakcie
  - Jednoduché na implementáciu
  - Malá odolnosť voči chybám, bottleneck

Distribuované zamykanie (2-fázový locking)

Primary-copy schéma

- Každý uzol je lock manager, zodpovedný za podmnožinu dát
- V prípade replikovaných dát, **práve jedna z kópií je primárna (t.j. jeden uzol spravuje primárnu kópiu)**
- O read-lock na dáta je možné žiadať ľubovoľného lock managera, ktorý má kópiu tých dát
- O write-lock treba žiadať primárneho lock managera, no ten musí žiadosť konzultovať s ostatnými lock managermi, ktorí majú kópiu dát
  - Menší bottleneck ako u centralizovaného algoritmu
  - Možnosť distribuovaného deadlocku

⟶ Primary-copy schéma je zhruba rovnako dobrá resp. zlá ako plne distribuovaná schéma

Distribuované zamykanie (2-fázový locking)

2 extremálne protokoly (plne distrib. schéma):

Distribuovaný 2-fázový ROWA protokol, Read-One-Write-All

- O read-lock na dáta je možné žiadať ľubovoľného lock managera, ktorý má kópiu tých dát
- O write-lock treba žiadať všetkých ostatných lock managerov, ktorí majú kópiu dát

Majoritný 2-fázový protokol

- O read- či write-lock na dáta treba získať súhlas **väčšiny** lock managerov, ktorí majú kópiu tých dát
- $2(n/2 + 1)$ správ pre lock, $(n/2 + 1)$ správ pre unlock
- Deadlock môže nastať aj pri zamykaní 1 záznamu: napr. každá z 3 transakcií môže vlastniť zámky na 1/3 kópií toho záznamu (tomuto sa dá vyhnúť, ak je dopredu dané poradie uzlov, ktorým sú posielané žiadosti o zámok)

Distribuované zamykanie (2-fázový locking)

Quorum protokol: Kompromis medzi ROWA a majoritným protokolom (plne distrib. schéma):

- Každému uzlu je priradená nejaká váha w_i, $w_i>0$
- Označme S sumu všetkých váh: $S=\sum w_i$,
- Nech Q_r (read quorum) a Q_w (write quorum) sú nezáporné čísla také, že $\mathbf{Q_r + Q_w > S}$ a zároveň $\mathbf{2\ Q_w > S}$
- Q_r a Q_w môžu byť dokonca rôzne pre rôzne záznamy

- Každý read musí získať zámok na toľkých kópiách, aby suma uzlov, ktoré tie kópie spravujú, bola aspoň Q_r
- Každý write musí získať zámok na toľkých kópiách, aby suma uzlov, ktoré tie kópie spravujú, bola aspoň Q_w

Distribuované zamykanie (2-fázový locking)

Quorum protokol: intuícia

- $Q_r + Q_w > S$

garantuje, že každé read a write quorum sa prekrývajú v aspoň jednom uzle (ktorý má aktuálnu hodnotu čítaného, resp. zapisovaného objektu). Navyše, nedá sa súčasne vytvoriť read a write quorum pre rovnaký objekt

- $2\ Q_w > S$

garantuje, že je nemožné vytvoriť súčasne dve rôzne write quora pre rovnaký objekt

Pripomína to pravidlá pre zamykanie, len na vyššej úrovni

Distribuovaná správa deadlockov

• Správa deadlockov je v distribuovanom systéme komplikovanejšia, lebo deadlocky môžu byť distribuované

Príklad: T1 beží na uzle 1, T2 beží na uzle 2

Uzol 1 (spravuje X)	**Uzol 2 (spravuje Y)**
write-lock1(X)	
w1(X)	
	write-lock2(Y)
	w2(Y)
	write-lock2(X)
write-lock1(Y)	

• T1 a T2 sú v deadlocku

• Lenže uzol 1 vie len to, že T2 čaká na T1, a uzol 2 vie len to, že T1 čaká na T2!

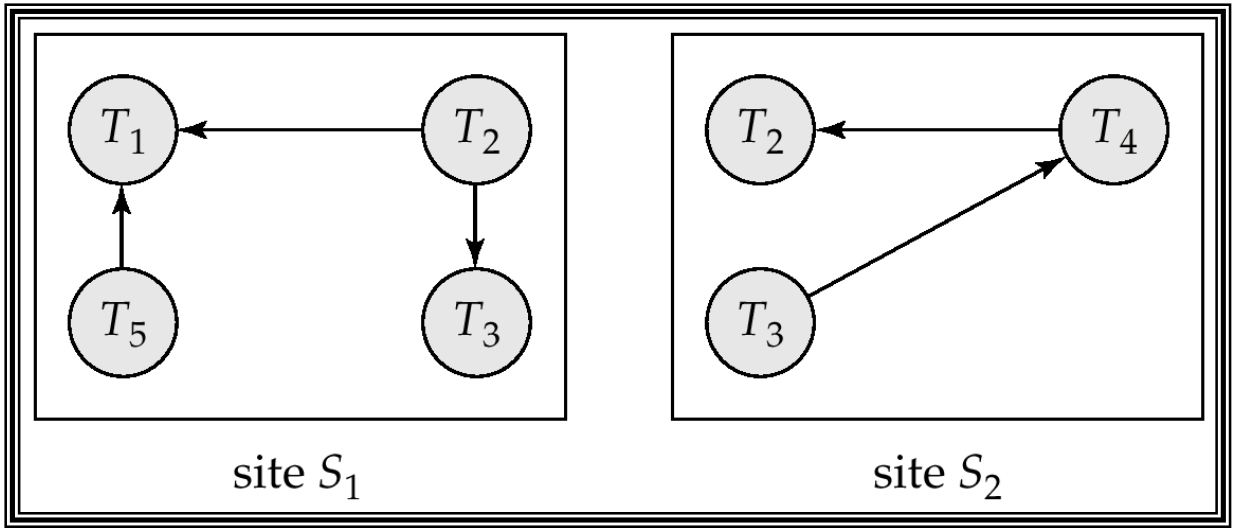
Distribuovaná správa deadlockov

- Riešenie: detekcia distribuovaných deadlockov a následný abort transakcií, ktoré spôsobujú deadlock
- Správa **lokálneho wait-for-grafu (WFG)** v každom uzle
- Distribuovaný protokol na konštrukciu **globálneho WFG**, ktorý kombinuje lokálne WFG
- Deadlock sa prejaví ako **cyklus v globálnom WFG**
- Globálny WFG konštruuje jeden uzol: **koordinátor**

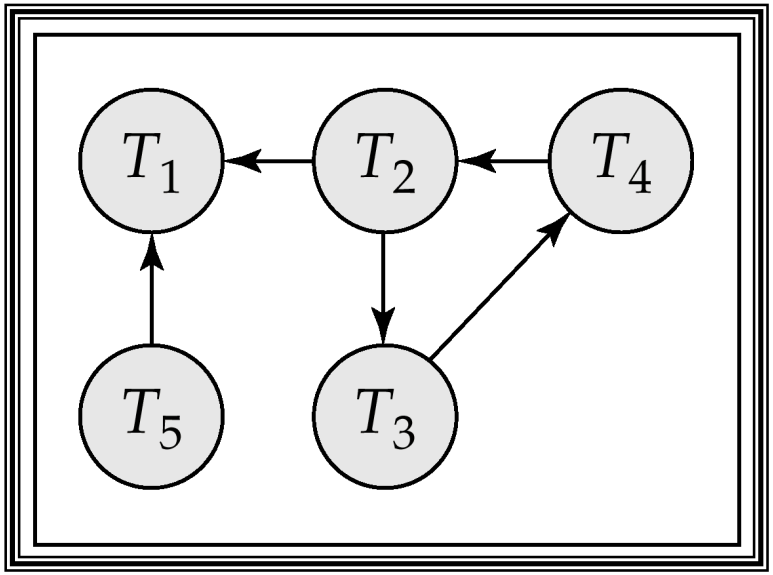
Distribuovaná správa deadlockov

Príklad

Lokálne WFG



Globálny WFG



Distribuovaná správa deadlockov

Protokol pre konštrukciu globálneho WFG

- Koordinátor pošle žiadosť o lokálne WFG
- Koordinátor skombinuje lokálne WFG do globálneho WFG a abortuje transakcie tak, aby odstránil cykly v globálnom WFG
- Abortovanie transakcií musí koordinátor oznámiť všetkým ostatným uzlom (treba skombinovať s atomickým commit protokolom)

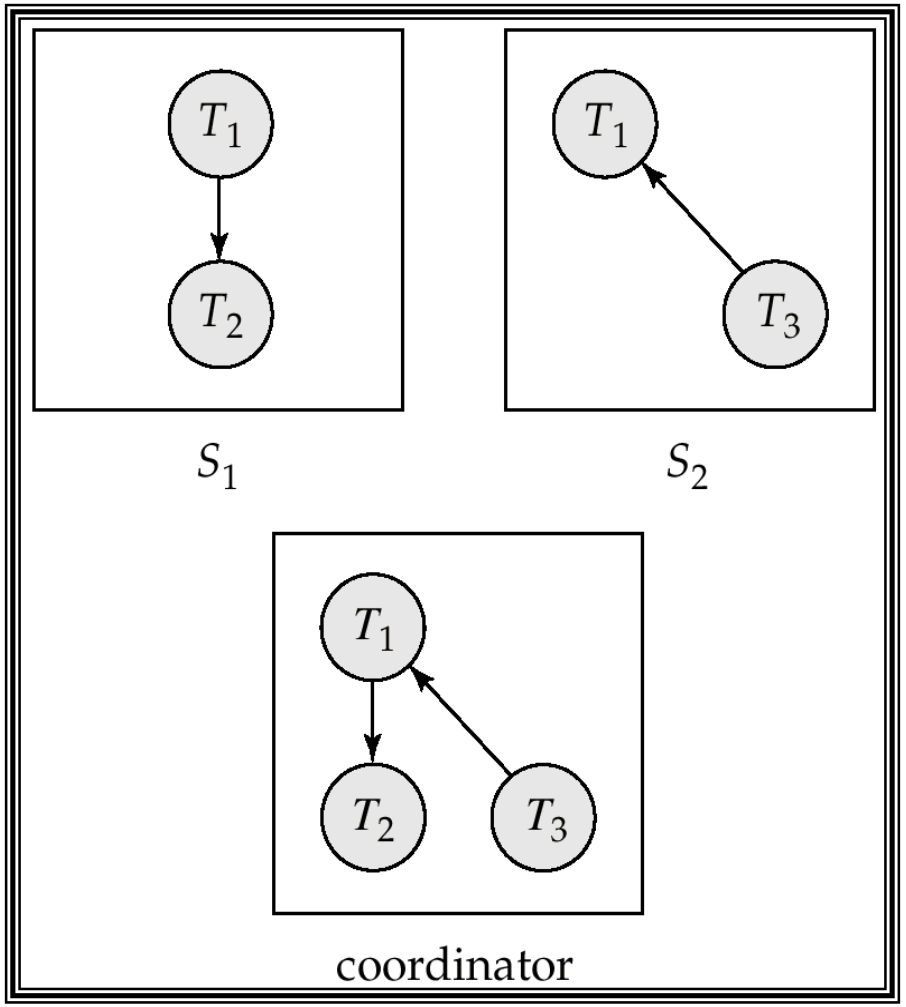
Problém:

- Koordinátor nepozná *skutočný* globálny WFG, len jeho *aproximáciu* (kvôli komunikačným oneskoreniam)
- Dôsledkom je detekcia cyklov, ktoré v skutočnosti neexistujú
- To znamená, že koordinátor niekedy abortuje transakciu, ktorú abortovať nemusí

Distribuovaná správa deadlockov

Príklad: Detekcia neexistujúcich cyklov

- T2 uvoľní zámok v uzle S1: hrana T1→T2 má byť zrušená
- T2 požiada o zámok, ktorý vlastní T3 v uzle S2: v uzle 2 pribudne hrana T2→T3
- Lenže koordinátor môže tieto 2 udalosti vidieť v opačnom poradí, ak mu S2 pošle svoj lokálny WFG skôr ako S1, vtedy detekuje cyklus T1 → T2→T3 →T1 v globálnom WFG a zbytočne abortuje niektorú z T1, T2, T3



Synchronizácia času (fyzické hodiny)

Pravidlá hry

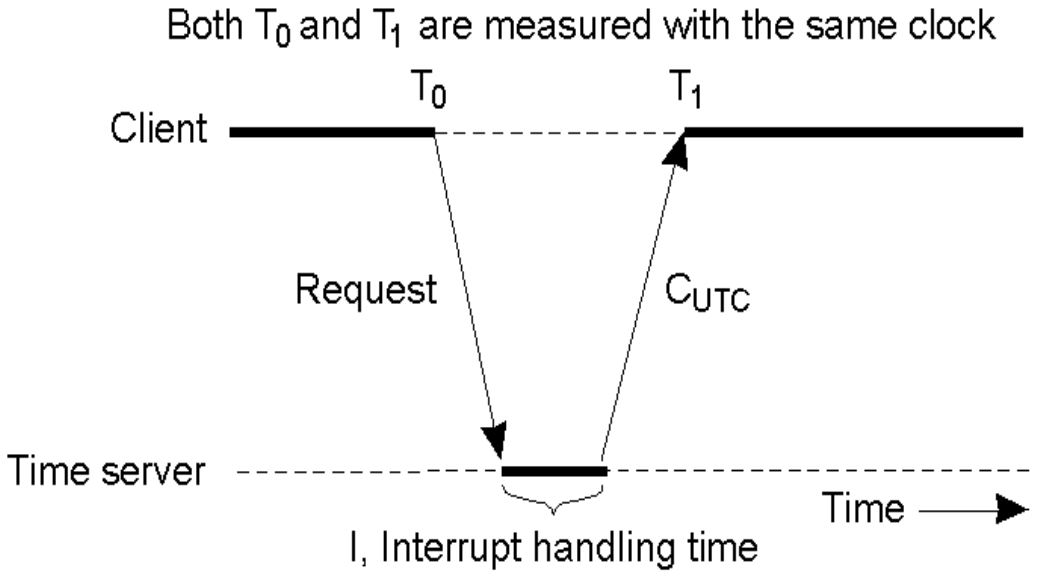
- Každý uzol má svoje lokálne hodiny
- Všetky hodiny idú vždy len dopredu, nikdy dozadu!
- Rôzne hodiny môžu ukazovať rôzne fyzické časy
- Rôzne hodiny môžu tikať rôzne rýchlo

Úlohou je hodiny všetkých uzlov synchronizovať, t.j. nastaviť (čo najpresnejšie) rovnaký čas vo všetkých uzloch

Synchronizácia času (fyzické hodiny)

Christianov protokol

• Jeden z uzlov je time-server, podľa ktorého sa všetky ostatné uzly synchronizujú



• Odhad komunikačného času: (T1 - T0) / 2, ak je I neznáme

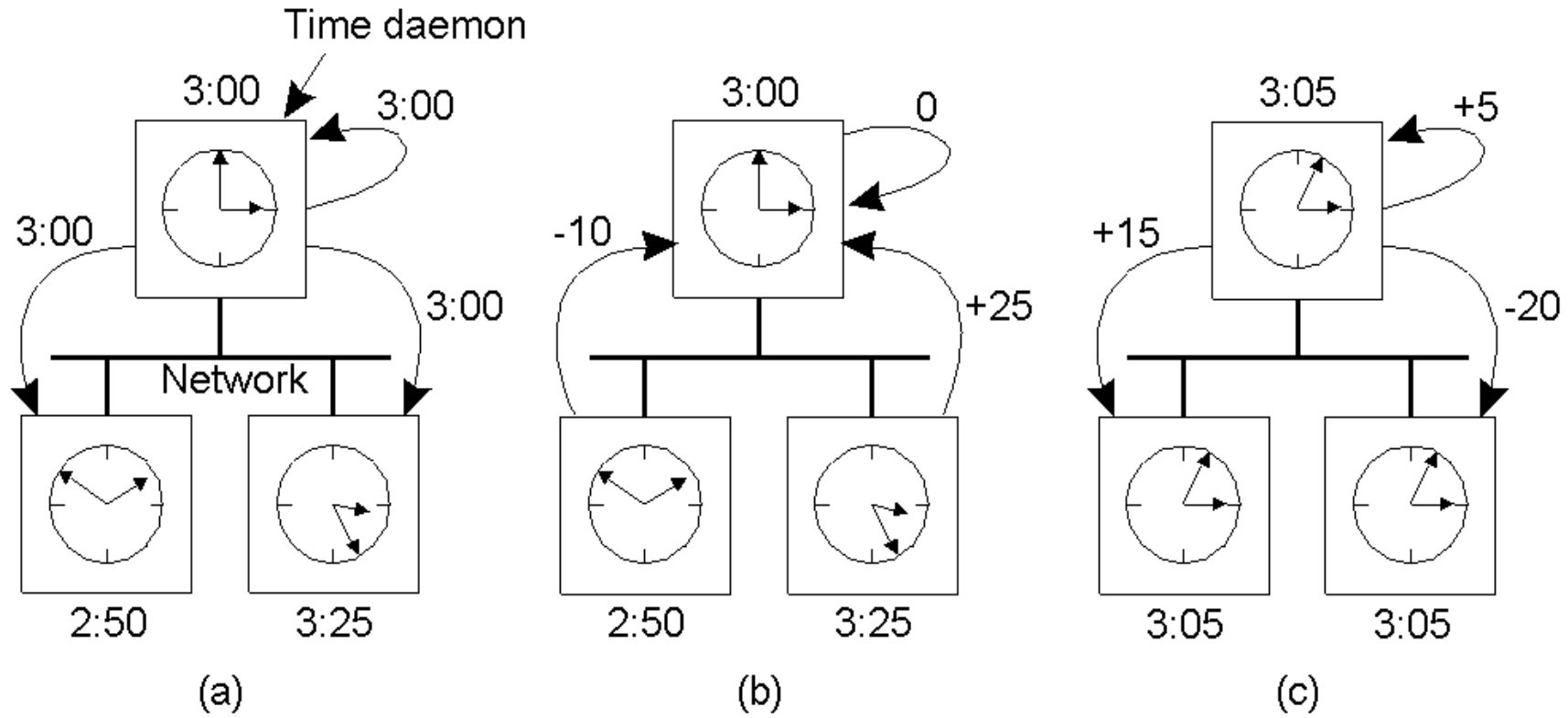
• Odhad komunikačného času: (T1 - T0 - I) / 2 ak je I známe

• Ak je čas time-servera C_{UTC} väčší ako čas klienta T1, tak klient posunie svoje hodiny dopredu na C_{UTC}+(T1 - T0 - I) / 2, inak klient na istý čas Δt **spomalí** svoje hodiny (hodiny musia ísť vždy vpred)

Synchronizácia času (fyzické hodiny)

Berkeley protokol

- Jeden z uzlov je time-server, ktorý sa synchronizuje podľa klientskych hodín (opak Christianovho prístupu)
- Time-server priemeruje svoje hodiny s hodinami klientov a koriguje všetky fyzické časy vrátane svojho



Synchronizácia času (logické hodiny)

Pravidlá hry

- Nie je dôležité aký čas ktoré hodiny ukazujú, dôležité je len poradie udalostí (napr. správa nebola poslaná skôr ako bola prijatá)
- Relácia A→B znamená “udalosť A sa stala pred udalosťou B”
- C(A): logický (nejaký) čas udalosti A

Úlohou je priradiť časové pečiatky $C(A_i)$ udalostiam A_i $(i=1,...,\infty)$ tak, aby platili nasledujúce požiadavky:

– Ak udalosť A_i nastala pred udalosťou A_j v jednom procese, tak potom $C(A_i) < C(A_j)$

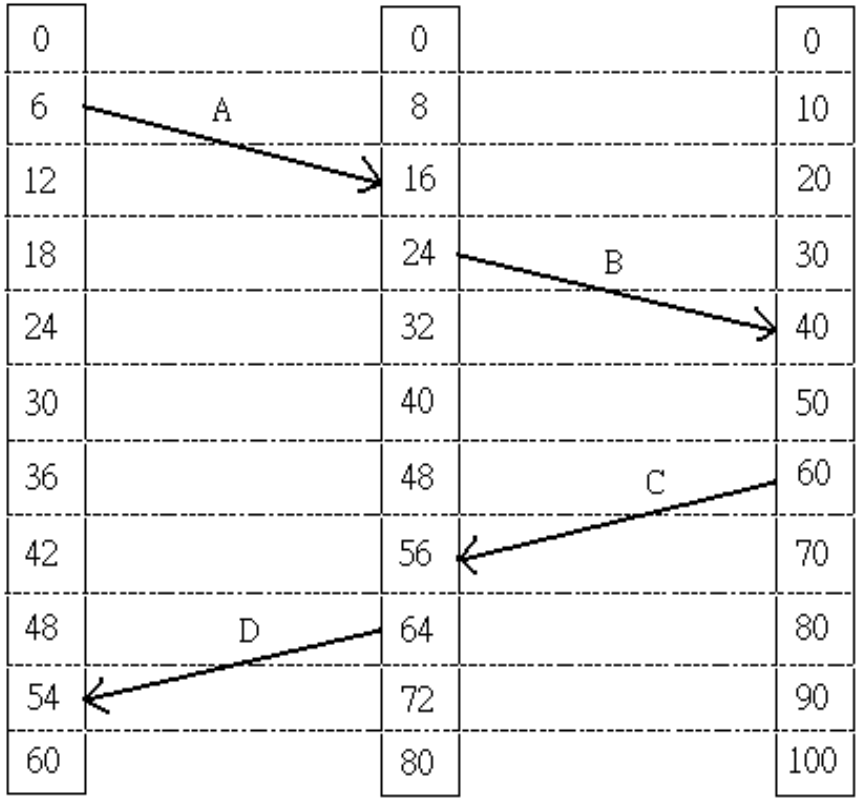
– Ak A_i znamená prijatie správy v nejakom procese a A_j znamená následné odoslanie správy z toho istého procesu, tak potom $C(A_i) < C(A_j)$

– Pre všetky pridelené časové pečiatky, ktoré popisujú sled udalostí, platí $C(A_i) \neq C(A_j)$

Synchronizácia času (logické hodiny)

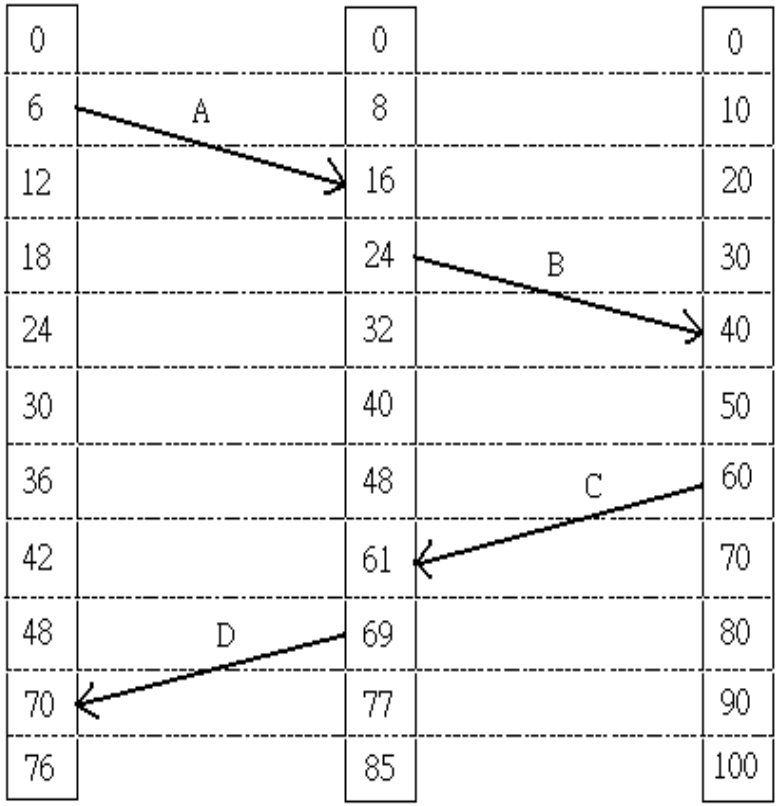
Lamportov protokol, príklad implementácie (korekcia lokálnych hodín podľa prichádzajúcej časovej pečiatky)

Fyzické hodiny v 3 procesoch



Časové pečiatky (nekorektné):
6, 16, 24, 40, 60, 56, 64, 54

Logické hodiny v 3 procesoch



Časové pečiatky (korektné):
6, 16, 24, 40, 60, 61, 69, 70

Synchronizácia času (logické hodiny)

Lamportov protokol, alternatívna implementácia (bez korekcie lokálnych hodín)

• Udalosti A, ktorá nastala v uzle S a každej správe odosielanej z uzla S sa priradí vektorová časová pečiatka $C(A)=[c_1, c_2,\ldots, \mathbf{c_S(A)}, \ldots, c_N]$, kde

– $c_S(A)$ je fyzický lokálny čas udalosti A v uzle S

– c_i je najvyšší fyzický čas uzla i ($i \neq S$), ktorý je známy uzlu S, t.j. najvyšší čas na i-tej pozícii vektorových časových pečiatok všetkých správ, ktoré uzol S prijal pred udalosťou A

– N je počet uzlov

• Sled udalostí je popísaný lexikografickým usporiadaním vektorových časových pečiatok

• Synchronizácia fyzických hodín nie je nutná (nutné je len to, aby fyzické hodiny všetkých uzlov išli vždy len dopredu)